POLITECNICO DI TORINO
Repository ISTITUZIONALE

Effective techniques for automatically improving the transition delay fault coverage of Self-Test Libraries

Original
Effective techniques for automatically improving the transition delay fault coverage of Self-Test Libraries / Cantoro, Riccardo; Garau, Francesco; Girard, Patrick; Kolahimahmoudi, Nima; Sartoni, Sandro; Reorda, Matteo Sonza; Virazel, Arnaud. - (2022), pp. 1-2. (Intervento presentato al convegno 2022 IEEE European Test Symposium tenutosi a Barcelona (SP) nel 23-27 Maggio 2022) [10.1109/ETS54262.2022.9810392].

Availability:
This version is available at: 11583/2969302 since: 2022-07-03T17:10:07Z

Publisher:
IEEE

Published
DOI:10.1109/ETS54262.2022.9810392

Terms of use:

This article is made available under terms and conditions as specified in the corresponding bibliographic description in the repository

Publisher copyright
IEEE postprint/Author's Accepted Manuscript

©2022 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collecting works, for resale or lists, or reuse of any copyrighted component of this work in other works.

(Article begins on next page)

25 February 2024

Effective techniques for automatically improving the transition delay fault coverage of Self-Test Libraries

Riccardo Cantoro*, Francesco Garau*, Patrick Girard†, Nima Kolahimahmoudi*,
Sandro Sartoni*, Matteo Sonza Reorda* and Arnaud Virazel†

*Department of Computer and Control Engineering
Politecnico di Torino
Turin, Italy

†LIRMM
University of Montpellier / CNRS
Montpellier, France

***Abstract*—In-field test of integrated circuits using Self-Test Libraries (STLs) is a widely used technique specifically suited to guarantee the processor's correct behavior during the operative lifetime, as mandated by functional safety standards such as ISO26262. Developing STLs for stuck-at faults requires significant manual efforts from test engineers, and targeting delay faults is even more challenging. In order to support this process, in this paper we propose a method to automate the creation of STLs targeting delay faults starting from existing STLs targeting stuck-at faults. The method is based first on identifying excited but not-observed transition delay faults and then adding suitable instructions able to detect them. Experimental results on a RISC-V processor show that the method can systematically detect a significant percentage of the target faults with reasonable computational effort and test code size increase.**

***Index Terms*—software-based self-test, software test libraries, in-field test, safety, functional test, delay faults**

I. Introduction

New semiconductor technologies are increasingly adopted in emerging applications. Such technologies, however, are extremely complex and sophisticated, leading to more frequent physical defects and reduced operative lifetime. Most of these defects are tested by targeting delay faults, such as transition delay faults (TDFs). Functional testing is an effective solution to tackle such issues. In the form of Software-Based Self-Test (SBST), functional testing [1] is based on the execution of STLs by the device under test (DUT). This approach has been proved effective both when processor cores [2]–[6] and peripherals [7]–[9] are tested. SBST is a desirable solution for *in-field testing*, i.e., when the device's reliability and safety has to be guaranteed throughout the operative lifetime, and can be used whenever compliance to standards such as the *ISO26262 standard* for automotive systems is required.

Developing STLs from scratch, however, is not trivial, as it requires a significant amount of manual effort. Moreover, understanding why certain faults are not detected is not always easy. The work in [10] moves the first step in classifying not-observed TDFs, defining some upper boundaries on how much the final TDF coverage can be increased.

In this paper, we propose an automatic and systematic methodology to increase the TDF coverage of STLs by detecting faults identified in [10] on complex pipelined processor cores, starting from a set of test programs devised for stuck-at faults (SAFs). The reported results show that it is possible to detect most of the TDFs (increasing the coverage by up to 15%) with a reasonable test time increase (from 15% to 22%) and computational effort. The article is organized as follows: in Section II related works are presented, while in Section III we describe the proposed approach. In Section IV we present the experimental results and, finally, in Section V we draw the conclusions.

II. Background

[10] introduces a study on TDFs in modern pipelined CPUs that have not been observed throughout the execution of STLs targeting SAFs. The article introduces *User Accessible Register* faults (UARs), whose effects reached registers that can be directly observed through instructions from the CPU's ISA, and *Hidden Register* (HRs) faults, whose effects reached registers non directly controllable. [10] provides some useful insights on what faults to target to improve the final fault coverage, and an upper bound on how much the final transition delay fault coverage can be improved. Strategies to detect them, however, are not given. Articles like [11], [12] focus on the improvement of available programs to obtain high fault coverage figures. These works show that methodologies for improving test programs can be successfully devised. Nonetheless, they are developed bearing the classical stuck-at fault model in mind.

III. Proposed Approach

This work focuses on the UAR fault category introduced in [10], defining some internal observation points, that do not require any hardware modification, to be used during simulation to further refine the topological analysis about fault effects.

In order to detect UAR faults, first we analyze the fault data base produced at the end of the fault simulation where internal observation points have been added. This is done to understand what UAR we have to work on, and what portion of the test program should be improved. In modern in-order pipelined CPUs there are instructions that take more than one